Using Streaming SIMD Extensions in a Fast DCT Algorithm for MPEG Encoding

Version 1.2

01/99

Order Number: 243651-002

Table of Contents

Revision History

Revision	Revision History	Date
1.2	FCS version.	01/99

References

The following documents are referenced in this application note, and provide background or supporting information for understanding the topics presented in this document.

1. Arai, Y., T. Agui, and M. Nakajima, (1988). A Fast DCT-SQ Scheme for Images, Trans IEICE, 71, pp. 1095-1097.
2. Feig E., and S. Winograd, (1992). Fast Algorithms for Discrete Cosine Transform, IEEE Trans. Signal Proc., 40, pp. 2174-2193.
3. Hung, A.C., and Thy Meng, (1994). A Fast Statistical Inverse Discrete Cosine Transform for Image Compression, SPIE/IS&Teletronic Imaging ,2187 , pp. 196-205.
4. Loeffler, C., A. Ligtenberg, and C. S. Moschytz, (1989). Practical Fast 1D DCT Algorithm with Eleven Multiplications, Proc. ICASSP 1989, pp. 988-991.
5. Winograd S. (1976). On Computing the Discrete Fourier Transform, IBM Res. Rep, RC-6291.
6. Lee, B. A New Algorithm to Compute the Discrete Cosine Transform, IEEE Trans. Signal Proc., Dec/84, pp. 1243-1245.
7. IEEE standard specification for the implementation of 8x8 iDCT IEEE std 1180-1990.
8. MPEG standard, Coding of Moving Pictures, ISO/IEC DIS 11172.
9. Mitchel, Pennebaker, Fogg, LeGall, (1997). MPEG video compression standard.
10. MMX™ technology application notes: Using MMX™ Instructions in a Fast iDCT Algorithm for MPEG Decoding; at `http://developer.intel.com/drg/mmx/appnotes/`.

1 Introduction

Streaming SIMD Extensions for the Intel® Architecture (IA) instruction set provide floating point single-instruction, multiple-data (SIMD) instructions. These instructions accelerate applications that rely heavily on operations that work on floating-point data (such as 3D graphics, real-time physics, and spatial audio).

This application note presents the implementation of a two-dimensional Discrete Cosine Transform (DCT) using Streaming SIMD Extensions. This type of transformation is widely used in image compression algorithms, most notably the DV, JPEG and MPEG. The document also includes examples of code that exploit the new integer instructions.

This document focuses on one DCT algorithm that provides efficient MPEG encoding. Three implementations of this algorithm are provided and discussed here:

- MMX™ technology code
- Code using Streaming SIMD Extensions – faster, with same accuracy as MMX technology code
- Code using Streaming SIMD Extensions – more accurate version

These three implementations can be used "as is" according to the guidelines presented in Section 3.1. However, many DCT algorithms exist. The reader is encouraged to consider the alternative ideas and issues presented in this document, since they have implications for other DCT algorithms.

1.1 MPEG Compression Method

The MPEG compression method has two phases, encoding and decoding. Multimedia images are first encoded, then transmitted, and finally decoded back into a sequence of images by the MPEG player. The encoding process follows these steps:

1. The input data is transformed using a Discrete Cosine Transform (DCT).
2. The resulting DCT coefficients are quantized.
3. The quantized coefficients are packed efficiently using Huffman tables and run length coding.

During the decoding process, the MPEG player reverses these steps by unpacking the coefficients, dequantizing them, and applying a 2D iDCT. To encode a high number of frames per second, this encoding process must be very fast. This document concentrates on the DCT component of the encoding process.

1.2 2D 8x8 DCT

At the heart of both intra and inter coding in MPEG is the DCT [10], which has certain properties that simplify coding models and make the coding efficient in terms of perceptual quality measures. For example, for a flat 8x8 block (all pixels with the same value), only the first (DC) coefficients of the DCT are nonzero. The DCT is a method of decomposing a block of data into a weighted sum of spatial frequencies.

The 8x8 two-dimensional DCT used in MPEG compression is defined as:

$$s(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7} c(u)c(v)\,f(v,u)\cos\frac{(2x+1)\pi u}{16}\cos\frac{(2y+1)\pi v}{16}$$

$$F(u,v) = c(u)c(v)\sum_{i=0}^{7}\sum_{j=0}^{7} Xij \cos\frac{(2i+1)\pi u}{16}\cos\frac{(2j+1)\pi v}{16}$$

where: Xij is the pixel value at [i,j] position, u and v are the horizontal and vertical frequency indices respectively, and the constants , c(u) and c(v) are given by:

$$c(k) = 1, iff k \neq 0, \frac{1}{2\sqrt{2}} otherwise$$

The solution to this equation contains 64 multiplications (8x8=64 multiplications with Xij) and 63 additions for each element, for a total of 4096 multiplications and 4032 additions for all 64 elements of the block..

The previous equation is equivalent to a summation over v, followed by a summation over u, as follows:

$$F(u,v) = \frac{c(u)}{2}\frac{c(v)}{2}\sum_{i=0}^{7}\cos[\frac{(2i+1)}{16}u\pi]\sum_{j=0}^{7} Xij\cos[\frac{(2j+1)}{16}v\pi]$$

$$c(k) = 1, iff k \neq 0, \frac{1}{\sqrt{2}} otherwise$$

This equation is the equivalent to applying a one-dimensional (1D) DCT eight times on each column, and then applying a 1D DCT on the rows of the result. Reversing this order by applying a 1D DCT on the rows first, and then on the columns, gives the same result.

2 AAN 2D DCT

The following sections explain the choice, functionality, and implementation of the AAN algorithm.

2.1 Background—Choosing a DCT Algorithm

Many algorithms have been proposed for efficient calculation of the 2D DCT. Some algorithms are based on efficient 1D DCTs [4]; some rely on direct analysis of the two-dimensional nature of DCT [2]; and others combine a 2D postscale with a very efficient 1D DCT [1]. Most fast 1D DCT and iDCT algorithms are variants of Lee's Fast DCT algorithm [6], or are based on variants of Winograd's FFT [5].

The following algorithms were evaluated for iDCT implementation using MMX instructions [11]:

- Feig's two-dimensional algorithm [2]
- The LLM algorithm [4]
- The AAN algorithm [1]

Although Feig's 2D algorithm [2] reduces the multiplication count, the cost of multiplication in MMX technology is small, so this reduction is not critical. Also, the irregular memory-access pattern of this algorithm is not conducive to efficient implementation using the MMX technology instruction set.

Both the LLM and AAN iDCT algorithms were implemented in MMX technology code. The LLM algorithm was implemented using accumulation in 32-bit elements, while the AAN algorithm used accumulation in 16-bit elements. The resulting LLM implementation was more accurate, conforming to the DCT IEEE standard [7]. However, the AAN implementation was much faster. Even though it was less accurate the AAN implementation was chosen in [10] and is presented in this document due to its speed.

2.2 AAN Algorithm

The AAN algorithm is a one-dimensional, postscaled DCT algorithm. First, a 1D DCT is applied on the eight input coefficients, which requires five multiplications and 29 additions for the transform [1]. Then the result is postscaled by eight postscale values, which requires eight multiplications. Although the 1D AAN algorithm is less efficient than other 1D algorithms (for example, LLM), when applied to the two-dimensional, 8x8 DCT, this algorithm takes only 64 multiplications for the postscale and 80 multiplications and 464 additions for the transform.

2.3 2D DCT AAN Algorithm Implemented with MMX™ Instructions and Streaming SIMD Extensions

The AAN implementation described in this document uses 16-bit data elements, so four variables can be processed in parallel using packed words. MMX instructions, which operate on packed words, read or store four words contiguously in memory. So, for an 8x8 matrix, half a row can be read or stored at one time. If the 1D DCT is applied to the columns of an 8x8 matrix, MMX instructions can operate on four columns at a time. Applying the 1D DCT to the rows of the matrix is more involved and less efficient.

The AAN algorithm is performed in four steps:

1. Perform a 1D DCT on the columns of the matrix.
2. Transpose the matrix.
3. Perform a second 1D DCT on the current columns of the matrix, which is equivalent to performing a DCT on the rows of the original matrix.
4. Postscale the output coefficients of the DCT on the columns of the transposed matrix.

These steps result in a transposed matrix, which would have to be transposed again to obtain the final result. These extra steps, of final postscale and output matrix transpose, might be easily implemented during the scan conversion (transpose during the zigzag) and during the quantization (postscale during the quantization). The cost of transposing the output is negligible, since the matrix is constructed from the Zig-Zag scan [9]. Therefore, the actual implementation follows these steps:

1. Perform a 1D DCT on the columns of the input matrix.
2. Transpose the matrix.
3. Perform a second 1D DCT on the columns of the transposed matrix.
4. Postscale the DCT coefficients of the transposed DCT matrix during the quantization
5. Transpose the DCT transformed matrix during the scan conversion (zig-zag).

The order of the transpose and postscale can be exchanged.

2.4 Applications for the 2D DCT

This DCT function is used in video compression algorithms: MPEG1, MPEG2, JPEG, DV etc.

This algorithm gives the main spatial compression on the intra- and inter-frames. The Streaming SIMD Extensions optimized version might be used on SW video encoders running on Streaming SIMD Extensions processors and later generations.

2.5 Implementing the 2D DCT Algorithm

The following sections describe several aspects of implementing the 2D DCT algorithm.

2.5.1 DCT Routine Interface

The DCT routine is implemented in 5 steps:

1. The assembler DCT routine is called from a C module. The routine gets two pointers: a pointer to an 8x8, 16-bit element input matrix and a pointer to an 8x8, 16-bit element output matrix. The input pointer is located in the `ESI` register and the output matrix pointer in the `EDI` register. The matrixes should be aligned on an 8-byte boundary. Each data element is actually a sign-extended 8-bit pixel (in intra block) or a sign-extended error term (in inter block).
2. The output matrix should be transposed. The routine uses the input matrix. Therefore, if the original input operands are needed (for example, in test mode), they must be copied before the call to the DCT routine.
3. The output matrix should be post scaled.
4. In the C code, there are two interface modules: `init_aan()` to initialize the post scale vector, and `postscale_transpose()` routine to produce standard DCT result that might be compared to other routine results.
5. In the optimized SIMD code, the `init_aan()` and `postscale_transpose()` might be collapsed in the quantizer and descan routines without additional performance penalty.

2.5.2 Optimization Considerations

A standard Pentium® II processor optimization technique is code rescheduling to exploit parallelism in an algorithm. Parallelism in the 2D DCT was approached from two directions, as illustrated in Figure 1:

- Within a single 8x8 DCT
- Between four 8x8 DCTs

In the first approach, data is accessed by rows within the matrix. In the second approach, data from the four matrixes is interleaved to enable efficient use of the MMX instructions.

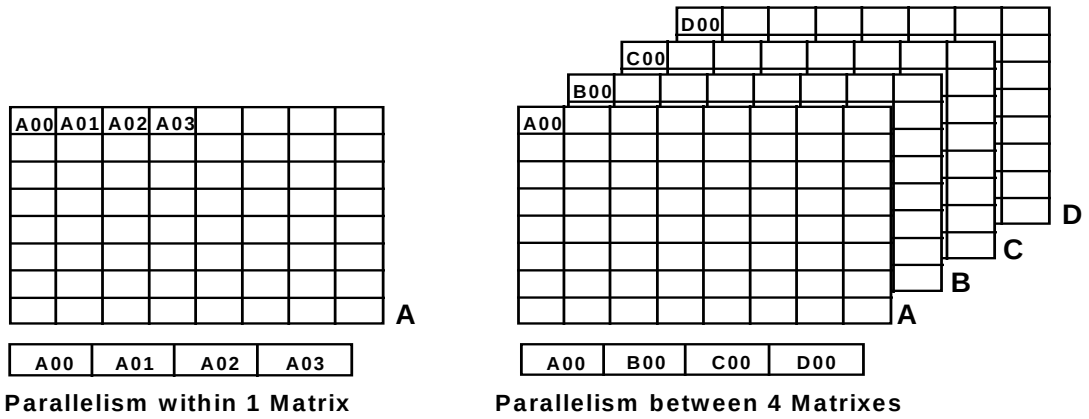

Figure 1. Single DCT vs. Four DCTs

The advantage of the single-DCT approach is that the interface to an MPEG encoder is simpler. Its disadvantages are:

- The matrix must be transposed in order to operate on several rows in parallel.
- To prevent overflow, packed shift instructions must be used. Since in a given register, the accuracy of the four data elements varies, the shift count is determined by the worst case among the four elements. This results in an extra loss of accuracy.

The advantages of the four-DCTs approach are:

- Matrix transposition is not required. To prevent overflow, packed shift instructions must still be used. However, since all the data elements in a register have the same accuracy, there is no extra loss of accuracy to accommodate the worst case among four elements.
- The DCT results are packed together according to the place in the matrix (0:63) so the quantization and the zig-zag can be done in SIMD.

The disadvantage of the four-DCT approach is that,

- To take advantage of the MMX instructions, the input data from the four matrixes must first be interleaved (see Figure 1).
- It also requires more register usage, so more temporal use of memory is needed.

The over-riding factor in our selection was the simplicity of implementation, because of which we chose the single-DCT approach. Instruction scheduling was done manually to ensure optimal performance.

In our manual scheduling of the instructions, register use was carefully considered. In most cases, we were able to keep the intermediate results in the registers; which resulted in minimal memory accesses for temporary storage.

2.5.3 DCT vs. iDCT Implementation

The current DCT implementation is based on the MMX™ technology implementation of the iDCT algorithm presented in [10]. The main differences between the two are:

- **Transpose implementation.** In iDCT, the output butterflies are symmetric, so an 8x8 transpose can be composed from 4x4 sub-transposes and the 4x4 transpose can be implemented on place. In DCT, the output is not symmetrical, so the transpose can not be implemented by 4x4 sub-transposes. The current solution is to use the Streaming SIMD Extensions, `pinsrw`, and implement the full transpose. This requires a second output matrix. Another possible solution would be to use a non-symmetrical, sub-transpose decomposition, using the inherent symmetries of the input butterflies of the second stage DCT (on rows). These ideas are not implemented in this note.
- **Statistical algorithms.** In iDCT, many iDCT 8x8 blocks have frequent zero elements, so using statistical algorithms that are optimized for the zero elements is natural and has potential speedup. In DCT the input is an 8x8-pixel matrix with no frequent zero elements. The only place with frequent zero elements would be after quantization.
- **Accuracy.** In iDCT, accuracy is more critical because the MPEG standard requires at most one grey level error. In DCT, there is no accuracy requirement.
- **Processing four blocks simultaneously.** This approach seems more beneficial in DCT than in iDCT because the DCT result is quantized, and the quantizer also may be implemented for four blocks. Another reason is that there are a lot of zero iDCT blocks, but fewer zero DCT blocks (the zero blocks are created only after the quantizer).

3 Performance

This section describes the performance gains achieved using Streaming SIMD Extensions.

3.1 Gains/Improvements

The main gain by using the Streaming SIMD Extensions over the MMX™ technology version is for the transform implementation. The second stage of the algorithm applies a transpose on the transformed columns to get the rows in column position. This enables SIMD operation on 4 rows together. In the MMX™ technology version, it is implemented directly, while in Streaming SIMD Extensions version, the `pinsrw` instruction creates the transposed words just before they are needed.

3.2 Accuracy vs. Speed

The MMX™ technology code and the fast Code using Streaming SIMD Extensions use packed signed multiply instructions, `pmulhw`, to implement the multiply operations.

The highest multiplier in the AAN DCT algorithm is 1.30. This number can be represented with 14-bit precision as a signed value, or 15-bit precision as an unsigned value. All other multipliers can be represented with 15-bit precision as signed values, or 16-bit precision as unsigned values.

The more accurate code using Streaming SIMD Extensions uses packed unsigned multiple instructions, `pmulhuw`, to increase accuracy.

3.3 Considerations

Even though the Pentium II processor is an out-of-order CPU, there is some potential for static scheduling optimizations, such as reducing source dependency.

Implementing the transpose with `pinsrw` is beneficial. However, reading data from memory as a word soon after writing the data as a qword causes a memory misalignment stall. To avoid this stall, copy from the MMX™ technology register, use the `pshufw` instruction to shift to the right word position for the first word, and merge two MMX™ technology registers for the other words.

Careful register allocation can minimize register spilling.

4 Conclusion

Using integer instructions in the DCT is beneficial in three ways:

- Implementing the transpose with `pinsrw` instructions.
- Using the `pshufw` instruction to avoid memory misalignment stalls.
- Gaining higher precision by using the `pmulhuw` instruction in the intermediate and post scale multiply operations.

For the same accuracy, the version is faster than the MMX™ technology version. It can also provide better accuracy using the packed unsigned multiply instruction, `pmulhuw`.

5 C Coding Example

The `\samples\DCT\` subdirectory contains the C and assembly code files discussed in this application note:

- `Aan_new.c` contains the C code implementation of the DCT AAN algorithm and the interface routines: `init_aan()` and `postscale_transpose()`.
- `Dctaan_mmx.asm` contains the MMX™ technology code implementation of the AAN DCT function.
- `Dctaan_kat.asm` contains the Code using Streaming SIMD Extensions implementation of the AAN DCT function.
- `Dctaan_kat_acc.asm` contains a more accurate version of Code using Streaming SIMD Extensions implementation of the AAN DCT function.
- `DCTstub.c` contains the C code implementation of operating the DCT routines and checking accuracy.